FINISHING WITH NUMBERS

LEARNING OBJECTIVES

- Do multiplication and division of binary values
- Use shifts to multiply and divide
- Binary fractions
- See how to represent values in a simple floating point format

BINARY MULTIPLICATION

- Because we only have two digits in binary, learning all of the times tables is pretty easy:
  - $0 \times 0 = 0$
  - $0 \times 1 = 0$
  - $1 \times 0 = 0$
  - $1 \times 1 = 1$

$$
\begin{array}{r}
100 \\
\times\ 11 \\
\hline
100 \\
1000 \\
\hline
1100
\end{array}
$$

- We then multiply in the same way we do in decimal.

YOU DO - BINARY MULTIPLICATION

```
   100        111        1111111
 × 111      × 111          × 101
-----      -----          -----
```

SHIFTS AND MULTIPLICATION

- As in decimal, multiplying by the power (2 or 10) is the same as moving all of the digits to the left and adding a zero on the right of the number.
  - $32_{10} \times 10_{10} = 320_{10}$ $1011_2 \times 10_2 = 10110_2$
- So in binary each bit shift to the left is the same as multiplying by 2.
- So multiplying by a power of 2 (e.g. $2^5 = 32$) is the same as shifting the binary number to the left 5 times.
- This is a way multiplication can be done in hardware, if we have LEFT SHIFTS (<<) and ADDITION (+) we can multiply.

DIVISION

- Division is a little more complicated.
  - https://andybargh.com/binary-division/
  - or its equivalent as an algorithm https://en.wikipedia.org/wiki/Division_algorithm#Integer_division_(unsigned)_with_remainder
    - You are not expected to know this technique in this course.
- But if we are dividing by a power of 2 then we can just shift the number to the right, once for each power. (Here we have a binary point rather than a decimal point.)
  - e.g. $1010111_2 \div 1000_2 = 1010.111_2$

BIGGER AND SMALLER

- What limits do we have on 16-bit numbers?
  - if using unsigned numbers we have 0 to 65535
  - if using two's complement we have -32768 to 32767
- Can we use those 16-bits in some other way?
- Floating point numbers use binary scientific notation
  - **Scientific notation**, base 10: $135000 = 1.35 \times 10^5$
  - Base 2: $3.25_{10} = 11.01_2 = 1.101 \times 2^1$

BINARY FRACTIONS

- In the same way that powers of two of positive numbers represent the binary digits for integers, powers of two of negative numbers represent the binary digits for fractions.
- $0.1001_2 = (1 \times 2^{-1}) + (0 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4}) = 0.5625$

$$\frac{1}{2} + \frac{1}{16} = \frac{8}{16} + \frac{1}{16} = \frac{9}{16}$$

- It is probably easiest to keep the answer as a fraction if you don't have a calculator

DECIMAL FRACTION TO BINARY

- With a decimal integer we use repeated division by 2 to get the binary equivalent
- With a decimal fraction we use repeated multiplication by 2 to get the binary equivalent
- e.g. 0.1875

ones or zeros	Answer so far
.	1875
0	375
0	75
1	5
1	0

read answer this way: 0.0011_2

EXAMPLE

0.24

ones or zeros	Answer so far
	.24
0	.48
0	.96
1	.92
1	.84
1	.68
1	.36
0	.72
1	.44
0	.88
1	.76
1	.52
1	.04
0	.08
0	.16
0	.32

Answer: $0.001111010111000\ldots_2$

YOUR TURN

Convert 0.375 to binary

Convert 0.4 to binary

FLOATING POINT

- If we take the idea of using some of the bits of a number to represent the exponent of a number as in scientific notation we can extend the range.
- There are many different floating point formats including several standard formats such as the IEEE 754 standard.
- We will use the one in the textbook.

BIGGEST

- In this format the "binary" point is assumed to be to the left of the first significant digit of the mantissa, so

0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1

is + $.111111111_2 \times 2^{+11111_2} = .111111111_2 \times 2^{31}$

$= 111111111_2 \times 2^{22}$

$= 511 \times 4{,}194{,}304$

or

$= 11111111100000000000000000000000_2$

$= 2{,}143{,}289{,}344$

- So the range in this format is from -2,143,289,344 to +2,143,289,344.
  - a lot bigger than -32768 to 32767 (but the gaps between numbers are bigger too)
  - we can also get very small numbers (with the exponent being negative)

CONVERTING TO FLOATING POINT

- Convert -120_{10} into this floating point format.
- First convert to binary as positive.
  - $120_{10} = 111\ 1000_2$
- Then normalize (move the binary point to the left of the first 1).
  - $111\ 1000 = .1111000 \times 2^7$
- Convert the exponent into binary
  - $7_{10} = 111_2$
- Then fill in the fields, sign = 1, sign of exponent = 0 so

1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1

In hex: F807

CONVERTING TO FLOATING POINT

- Convert 20.75_{10} into this floating point format.
- First convert to binary as positive.
  - $20.75_{10} = 10100.11_2$
- Then normalize (move the binary point to the left of the first 1).
  - $10100.11 = .1010011 \times 2^5$
- Convert the exponent into binary
  - $5_{10} = 101_2$
- Then fill in the fields, sign = 0, sign of exponent = 0 so

0	1	0	1	0	0	1	1	0	0	0	0	0	1	0	1

In hex:5305

GOING THE OTHER WAY

- What does $C102_{16}$ represent?
- First write the binary.

1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0

- The number is negative, the mantissa is 0.100000100 the exponent is positive, the exponent is 2
  - $-0.1000001_2 \times 2^2 = -10.00001_2$
- which is -2.03125

YOU DO

- Convert 1 into the floating point format
- Convert -1024 into the floating point format
- Convert the floating point number represented by 0x5004 in this format into decimal (0x in front a number is a common programming language way of meaning this is hexadecimal)
- Extra for experts - what is the smallest (closest to zero) number which can be represented in this floating point format?

REVISION

- Convert -0.5 into the text book's floating point format. Write the answer in hexadecimal
- Convert the floating point number represented by 0x5004 in this format into decimal (0x or x in front a number is a common programming language way of meaning this is hexadecimal)
- Convert 0xF00F from the floating point format into decimal.

1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1

CHARACTER ENCODING

- Originally there were many competing ways to encode the characters of human languages into binary.
- Now we map characters onto binary numbers in a standard way (sort of)
  - **ASCII** (8-bit numbers for each character)
    - American Standard Code for Information Interchange
  - **Unicode** (different related standards, e.g. UTF-8, UTF-16, UTF-32) (Unicode Transformation Format)
    - Universal Coded Character Set (UCS)

ASCII

- Developed from earlier codes
- 7-bit teleprinter code (there are 8-bit extensions)
- 128 different values
  - first 32 are control codes (to control printers and can be used for other purposes) also 0x7F (DEL)
  - all UPPERCASE letters come before lowercase

Public Domain, https://commons.wikimedia.org/w/index.php?curid=225986

dec	oct	hex	ch	dec	oct	hex	ch	dec	oct	hex	ch	dec	oct	hex	ch
0	0	00	**NUL** (null)	32	40	20	(space)	64	100	40	**@**	96	140	60	**`**
1	1	01	**SOH** (start of header)	33	41	21	**!**	65	101	41	**A**	97	141	61	**a**
2	2	02	**STX** (start of text)	34	42	22	**"**	66	102	42	**B**	98	142	62	**b**
3	3	03	**ETX** (end of text)	35	43	23	**#**	67	103	43	**C**	99	143	63	**c**
4	4	04	**EOT** (end of transmission)	36	44	24	**$**	68	104	44	**D**	100	144	64	**d**
5	5	05	**ENQ** (enquiry)	37	45	25	**%**	69	105	45	**E**	101	145	65	**e**
6	6	06	**ACK** (acknowledge)	38	46	26	**&**	70	106	46	**F**	102	146	66	**f**
7	7	07	**BEL** (bell)	39	47	27	**'**	71	107	47	**G**	103	147	67	**g**
8	10	08	**BS** (backspace)	40	50	28	**(**	72	110	48	**H**	104	150	68	**h**
9	11	09	**HT** (horizontal tab)	41	51	29	**)**	73	111	49	**I**	105	151	69	**i**
10	12	0a	**LF** (line feed - new line)	42	52	2a	*	74	112	4a	**J**	106	152	6a	**j**
11	13	0b	**VT** (vertical tab)	43	53	2b	**+**	75	113	4b	**K**	107	153	6b	**k**
12	14	0c	**FF** (form feed - new page)	44	54	2c	**,**	76	114	4c	**L**	108	154	6c	**l**
13	15	0d	**CR** (carriage return)	45	55	2d	**-**	77	115	4d	**M**	109	155	6d	**m**
14	16	0e	**SO** (shift out)	46	56	2e	**.**	78	116	4e	**N**	110	156	6e	**n**
15	17	0f	**SI** (shift in)	47	57	2f	**/**	79	117	4f	**O**	111	157	6f	**o**
16	20	10	**DLE** (data link escape)	48	60	30	**0**	80	120	50	**P**	112	160	70	**p**
17	21	11	**DC1** (device control 1)	49	61	31	**1**	81	121	51	**Q**	113	161	71	**q**
18	22	12	**DC2** (device control 2)	50	62	32	**2**	82	122	52	**R**	114	162	72	**r**
19	23	13	**DC3** (device control 3)	51	63	33	**3**	83	123	53	**S**	115	163	73	**s**
20	24	14	**DC4** (device control 4)	52	64	34	**4**	84	124	54	**T**	116	164	74	**t**
21	25	15	**NAK** (negative acknowledge)	53	65	35	**5**	85	125	55	**U**	117	165	75	**u**
22	26	16	**SYN** (synchronous idle)	54	66	36	**6**	86	126	56	**V**	118	166	76	**v**
23	27	17	**ETB** (end of transmission block)	55	67	37	**7**	87	127	57	**W**	119	167	77	**w**
24	30	18	**CAN** (cancel)	56	70	38	**8**	88	130	58	**X**	120	170	78	**x**
25	31	19	**EM** (end of medium)	57	71	39	**9**	89	131	59	**Y**	121	171	79	**y**
26	32	1a	**SUB** (substitute)	58	72	3a	**:**	90	132	5a	**Z**	122	172	7a	**z**
27	33	1b	**ESC** (escape)	59	73	3b	**;**	91	133	5b	**[**	123	173	7b	**{**
28	34	1c	**FS** (file separator)	60	74	3c	**<**	92	134	5c	****	124	174	7c	**|**
29	35	1d	**GS** (group separator)	61	75	3d	**=**	93	135	5d	**]**	125	175	7d	**}**
30	36	1e	**RS** (record separator)	62	76	3e	**>**	94	136	5e	**^**	126	176	7e	**~**
31	37	1f	**US** (unit separator)	63	77	3f	**?**	95	137	5f	**_**	127	177	7f	**DEL** (delete)

THINGS TO NOTE ABOUT ASCII

- Designed to help with sorting (unlike earlier codes)
  - First 32 characters - control codes
  - Digits '0' to '9' have hexadecimal encodings 0x30 to 0x39
  - Uppercase letters 'A' to 'Z' are in order with encodings 0x41 to 0x5A
  - Lower case letters 'a' to 'z' have encodings 0x61 to 0x7A (hence only one bit difference between an upper and a lowercase letter)
- Symbols are scattered around (for reasons - see https://en.wikipedia.org/wiki/ASCII#Internal_organization), and the keyboard you use on your computer has all of those symbols (and possibly no others)
- ASCII has been superseded, mostly by UTF-8 which keeps the ASCII code for 8-bit values from 0x00 to 0x7F